# 第5章　字符串与正则表达式

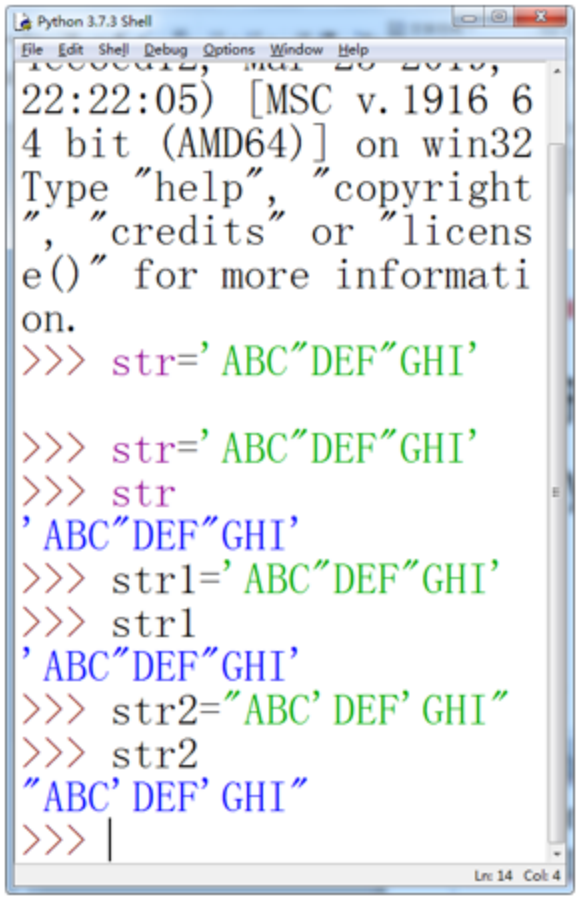
# 内容提要

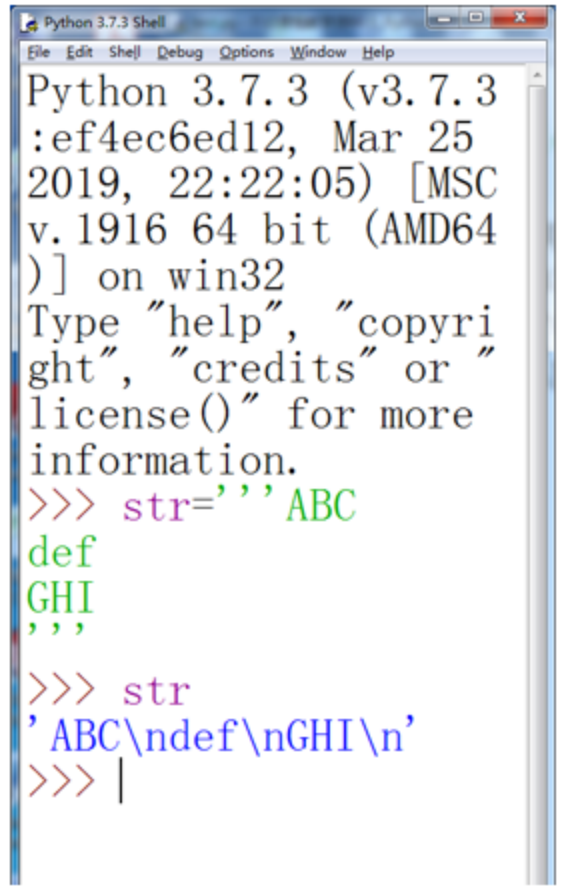
- 字符串
- 正则表达式

# 1. 字符串

字符串的创建

Python中的字符串是一个有序的字符序列，可以用对单引号、双引号或三引号表示。



# 1. 字符串

- 字符串的基本操作
  - 元素读取：串名[索引]

```
>>> s="Python"
>>> s[0]
'P'
```

  - 求串长：len(串名)

```
>>> len(s)
6
```

  - 成员运算：字符串1 [not]in 字符串2

```
>>> "tho" in s
True
```

```
>>> "Tho" not in s
True
```

# 1. 字符串

- 字符串的基本操作
  - 串连接：支持利用空格、加号等将多个字符串连接成一个新的字符串

```
>>> s="ab"+"cd"+"ef"
>>> s
'abcdef'
```

```
>>> s="AB" "CD"
>>> s
'ABCD'
```

```
>>> '123'*3
'123123123'
```

# 1. 字符串

- 字符串的基本操作
  - 串分片：串名[start:end:step]

说明：start 表示子串的起始位置，end表示子串的终止位置（不含end对应的字符），step表示步长。

```
>>> s='Python'
>>> s[1:4]
'yth'
>>> s[2:]
'thon'
>>> s[:-1]
'Pytho'
```

```
>>> s[0:7:2]   #步长为2，所以取第1、3、5位置上的字符组成子串
'Pto'
>>> s[::-1]    #串逆置
'nohtyP'
```

# 1. 字符串

- 字符串的基本操作
  - 串转换：可以用内置函数str()将数字转换成字符串

```
>>> str(5.14)
'5.14'
```

# 1. 字符串

- 字符串的基本操作
  - 串比较：字符串比较的依据是相应字符的ASCII码值大小，关系成立返回 True，否则返回False。

```
>>> "abc">"abC"
True
>>> "abc">"abcd"
False
```

# 1.字符串

- 字符串处理函数

| 函　数 | 功能描述 |
| --- | --- |
| len(s) | 返回字符串、列表、字典、元组等长度 |
| str(s) | 返回任意类型 s 的字符串形式 |
| chr(x) | 返回 Unicode 编码 x 对应的单字符 |
| ord(s) | 返回单字符 s 对应的 Unicode 编码 |
| hex(x) | 返回整数 x 对应十六进制数的小写形式字符串 |
| oct(x) | 返回整数 x 对应八进制数的小写形式字符串 |

```
>>> chr(65)
'A'
>>> ord('A')
65
```

```
>>> hex(65)
'0x41'
>>> oct(65)
'0o101'
```

# 1. 字符串

- 字符串处理方法

**str类中封装了很多字符串处理函数可以通过“对象名.成员函数名( )”的方式使用。**

| 方法 | 功能描述 |
| --- | --- |
| str.strip() | 删除 str 串中两端空格后形成新串 |
| str.isalpha() | 判断 str 串中是否全是字母，返回逻辑值 |
| str.isdigit() | 判断 str 串中是否全是数字，返回逻辑值 |
| str.upper() | 将 str 串中字母全部转换成大写字母后形成新串 |
| str.find(substr,[start,[,end]]) | 定位子串 substr 在 str 中第一次出现的位置，start 和 end 用于限定定位的范围 |
| str.replace(old,new,[,count]) | 用字符串 new 替换 str 中的 old，可选参数 count 表示被替换的子串个数 |
| str.split([sep]) | 以 sep 为分隔符，将 str 分隔成一个列表，默认分隔符是空格 |
| str.join(sequence) | 用于将序列 sequence 中的元素以指定的字符（str）连接生成一个新的字符串 |

# 1. 字符串

- 字符串处理方法示例

```
>>> str="  ABC  "
>>> str.strip()
'ABC'
>>> str
'  ABC  '
>>> str.lstrip()
'ABC  '
>>> str.rstrip()
'  ABC'
```

```
>>> str="abcDEF"
>>> str.isalpha()
True
>>> str.upper()
'ABCDEF'
>>> str.isdigit()
False
>>> str.find("DE")
3
>>> str.replace("DE","de")
'abcdeF'
>>> str
'abcDEF'
```

# 1. 字符串

- 字符串处理方法示例

```
>>> str="ABC,DEF,GHI"
>>> str.split(',')
['ABC', 'DEF', 'GHI']
>>> list1=['A','B','C']
>>> '#'.join(list1)
'A#B#C'
>>> ''.join(list1)
'ABC'
```

```
>>> str="ABABABCD"
>>> str.count("AB")
3
```

# 1. 字符串

- 字符串的format()方法

基本语法是通过 {} 和 : 来代替以前的 % 。

(1) format 函数可以接受不限个参数，位置可以不按顺序。

(2) 可以设置参数

(3) 可以向 str.format() 传入对象

(4) 数字格式化（详见教材表5.3）

# 1. 字符串

- 字符串format()方法示例

```
# 不设置指定位置，按默认顺序
>>> "{}#{}".format("hello", "world")
'hello#world'

# 设置指定位置
>>> "{1} {0} {1}".format("hello", "world")
'world hello world'
```

# 1. 字符串

- 字符串综合案例

【例5-1】 从键盘输入一行字符，统计并输出其中英文字符（大小写分开）、数字和其它字符的个数。

分析：对输入的字符串，根据各个字符的ASCII码判断其类型。数字0~9对应的ASCII码：48~57，大写字母A~Z对应的ASCII码：65~90，小写字母a~z对应的ASCII码：97~122。

【例5-1】exp5-1.py

# 1. 字符串

- 字符串综合案例

【例5-2】统计一行字符串中英文单词的个数，设单词间由空格间隔。

分析：将连续的不含空格的字符串都当成单词，所以应先将字符串首尾的空格删除，然后从第一个非空格字符开始，至下一个空格前为第一个单词，再从下一个非空格字符开始统计第二个单词，依此类推。

【例5-2】exp5-2.py

# 2. 正则表达式

- 概念

正则表达式使用单个字符串来描述、匹配一系列匹配某个句法规则的字符串。

例：Windows下，输入“file?.doc”的搜索模式，将会查找到下列所示的文件：file1.doc, file2.doc, file3.doc。

输入“file*.doc”，则会找到以下示例文件：file.doc, file1.doc, file2.doc, , file12.doc, filexyz.doc。

# 2. 正则表达式

- 普通字符正则表达式

用英文字母、数字和标点符号来构成一个字符串的模式。

```
import re                                  #导入re模块
key = "pear apple orange"                  #要匹配的文本
p = r"apple"                               #正则表达式规则
pattern = re.compile(p)                    #编译这段正则表达式
matcher = re.search(pattern,key) #在指定字符串中搜索符合正则表达式的部分
print(matcher.group(0))                     #打印出来
print(re.search(pattern,key).span()) #打印出匹配上字符串的位置范围
```

```
apple
(5, 10)
```

# 2. 正则表达式

## 常用正则表达式

| 实例 | 含　义 |
| --- | --- |
| python | 匹配“python”。 |
| [Pp]ython | 匹配“Python”或“python”。 |
| Rub[y\|e] | 匹配“Ruby”或“Rube”。 |
| [aeiou] | 匹配任意一个元音字母。 |
| [^aeiou] | 匹配除 a、e、i、o、u 以外的所有字符。 |
| [0-9] | 匹配任何数字，类似于[0123456789]。 |
| [a-z] | 匹配任何小写字母。 |
| [A-Z] | 匹配任何大写字母。 |
| [a-zA-Z0-9] | 匹配任何字母和数字。 |
| + | 匹配前面的子表达式一次或多次。 |
| . | 匹配除“\n”外的任何单个字符。 |
| \d | 匹配一个数字字符，等价于[0-9]。 |
| \D | 匹配一个非数字字符，等价于[^0-9]。 |
| \w | 匹配包括下划线的任何单词字符，等价于’[a-zA-Z0-9]’。 |
| \W | 匹配任何非单词字符，等价于’[^a-zA-Z0-9]’。 |

# 2. 正则表达式

## 常用正则表达式示例

```
>>> import re                                          #导入re模块
>>> print(re.search(r'Pytho.','I love Python.com'))
<re.Match object; span=(7, 13), match='Python'>  # span表示以元组形式返回匹配到的范围
```

```
>>> print(re.search(r'\d\d\d','I love 123 FishC.com'))
<re.Match object; span=(7, 10), match='123'>
```

```
>>> print(re.search(r'\D+','123abc456'))
<re.Match object; span=(3, 6), match='abc'>
```

```
>>> print(re.search(r'[aeiouAEIOU]','I love FishC.com'))  #匹配一个元音字符
<re.Match object; span=(0, 1), match='I'>
```

```
>>> print(re.search(r' [^aeiouAEIOU] ', 'Idon\'t love Python')) #匹配一个非元音字符
<re.Match object; span=(1, 2), match='d'>
```

```
>>> print(re.search(r'[0-9]','I love 123 Python.com'))
<re.Match object; span=(7, 8), match='1'>
```

# 2. 正则表达式

## 特殊字符正则表达式

| 特殊字符 | 含　　义 |
| --- | --- |
| ^ | 匹配输入字符串的开始位置，如果设置了 re.MULTILINE 标志，^ 也匹配换行符之后的位置。在方括号中使用时，表示不接受该字符集合。要匹配^字符本身，需使用\^ |
| $ | 匹配输入字符串的结束位置，如果设置了 re. MULTILINE 标志，$也匹配换行符之前的位置。要匹配$字符本身，需使用\$ |
| () | 匹配圆括号中的正则表达式，或者指定一个子组的开始和结束位置，注：子组的内容可以在匹配之后被“\数字”再次引用。要匹配圆括号本身，需\(和\) |
| * | 匹配前面的子表达式零次或多次，等价于{0,}，要匹配*字符本身，需使用* |
| + | 匹配前面的子表达式一次或多次，等价于{1,}，要匹配+字符本身，需使用\+ |
| ? | 匹配前面的子表达式零次或一次，等价于{0,1}，要匹配?字符本身，需使用\? |

# 2. 正则表达式

## 特殊字符正则表达式

| 特殊字符 | 含　义 |
| --- | --- |
| […] | 字符类，匹配中括号所包含的任意一个字符。要匹配[字符本身，需使用\[<br>(1)连字符 - 如果出现在字符串中间表示字符范围描述，如果出现在首位则仅作为普通字符；<br>(2)特殊字符仅有反斜线 \ 保持特殊含义，用于转义字符。其它特殊字符如 *、+、？等均作为普通字符匹配；<br>(3)脱字符 ^ 如果出现在首位则表示匹配不包含其中的任意字符，如果 ^ 出现在字符串中间就仅作为普通字符匹配。 |
| \ | (1)将一个普通字符变成特殊字符，例如 \d 表示匹配所有十进制数字，\w 表示匹配所有字母和数字字符， \D 表示匹配任何非十进制数字<br>(2)解除元字符的特殊功能，例如 \. 表示匹配点号本身<br>(3)引用序号对应的子组所匹配的字符串 |
| {M,N} | M 和 N 均为非负整数，其中 M <= N，表示前边的 RE 匹配 M ~ N 次。<br>(1){M,} 表示至少匹配 M 次<br>(2){,N} 等价于{0,N}<br>(3){N} 表示需要匹配 N 次 |
| \| | A \| B，表示匹配正则表达式 A 或者 B，要匹配\|字符本身，需使用\| |

# 2. 正则表达式

## 特殊字符正则表达式示例

```
>>> import re                                          #导入re模块
>>> print(re.search(r‘^[0-9]+abc$’,‘123abc’))  #从字符串开始位置，匹配由一到多个数字后连接“abc”字符串
<re.Match object; span=(0, 6), match='123abc'>
```

```
>>> print(re.search(r'ab{3}c','abbbbbcdefg'))    #字符’b’多于3次则匹配失败
None
>>> print(re.search(r'ab{3,10}c','abbbbbcdefg')) #指定重复次数范围则匹配成功
<re.Match object; span=(0, 7), match='abbbbbc'>
```

```
>>> print(re.search(r'\d{3}-\d{3}-\d{4}','800-555-1212'))  #匹配电话号码
<re.Match object; span=(0, 12), match='800-555-1212'>
>>> print(re.search(r'\w+@\w+.com','xxx@163.com'))    #匹配电子邮件地址
<re.Match object; span=(0, 11), match='xxx@163.com'>
```

```
>>> print(re.search(r'(([01]{0,1}\d{0,1}\d|2[0-4]\d|25[0-5])\.){3}([01]{0,1}\d{0,1} \d|2[0-4]\d|25[0-5])','192.168.1.1'))
                                                    #匹配IP地址
<re.Match object; span=(0, 11), match='192.168.1.1'>
```

# 2. 正则表达式

## 非打印字符正则表达式

| 字符 | 含　义 |
| --- | --- |
| \cx | 匹配由 x 指明的控制字符。例如，\cM 匹配一个 Control-M 组合键或回车符。x 的值必须为 A-Z 或 a-z 之一。否则，将 c 视为一个原义的'c'字符。 |
| \f | 匹配一个换页符。等价于\x0c 和\cL。 |
| \n | 匹配一个换行符。等价于\x0a 和\cJ。 |
| \r | 匹配一个回车符。等价于\x0d 和\cM。 |
| \s | 匹配任何空白字符，包括空格、制表符、换页符等。等价于[\f\n\r\t\v]。 |
| \S | 匹配任何非空白字符。等价于 [^\f\n\r\t\v]。 |
| \t | 匹配一个制表符。等价于\x09 和\cI。 |
| \v | 匹配一个垂直制表符。等价于\x0b 和\cK。 |

# 2. 正则表达式

- re模块

Python语言中，可以使用re模块的内置函数来处理正则表达式。re模块主要包括编译正则表达式的函数和各种匹配函数。为便于理解，以下函数语法格式做了简化。

（1）compile()函数

功能：编译正则表达式。

语法格式：compile(source, filename, mode)

参数说明：

- source：字符串
- filename：代码文件名称，如果不是从文件读取代码则传递一些可辨认的值
- mode：指定编译代码的种类。可以指定为exec, eval和single

# 2. 正则表达式

- 示例

```
>>> import re
>>> pattern = re.compile(r'([a-z]+) ([a-z]+)', re.I)   # re.I 表示忽略大小写
                                                        # +表示1或多次
>>> m = pattern.match('Hello World Wide Web')
    #按正则表达式匹配字符串，注意中间有一个空字符，故只能匹配两个单词
>>> m.group()
'Hello World'
```

# 2. 正则表达式

- re模块

（2）match()函数

功能：从字符串的起始位置匹配一个模式，如果不是起始位置匹配成功的话，match()就返回none。若匹配成功则返回匹配上的对象实例。

语法格式：re.match(pattern, string)

参数：

- pattern:匹配的正则表达式
- string:要匹配的字符串

# 2. 正则表达式

- 示例

```
>>> import re
>>> print(re.match(r'How', 'How are you').span())  # 在起始位置匹配
(0, 3)
>>> print(re.match(r'are', 'How are you'))  # 不在起始位置匹配
None
```

# 2. 正则表达式

- re模块

（3）search()函数

功能：扫描整个字符串并返回第一个成功的匹配，若匹配成功则返回匹配上的对象实例。否则返回none。

语法格式：re. search (pattern, string)

参数说明：同match ()函数

# 2. 正则表达式

- 示例

```
>>> content = 'Hello 123456789 Word'
>>> print(re.search('(\d+)', content))#匹配字符串中的数字
<re.Match object; span=(6, 15), match='123456789'>
```

# 2. 正则表达式

- re模块

（4）findall()函数

功能：在字符串中查找所有符合正则表达式的字符串，并返回这些字符串的列表。否则返回none。

语法格式：re. findall (pattern, string)

参数说明：同match ()函数

# 2. 正则表达式

- 示例

```
>>> print(re.findall("a|b", "abcabc"))
 # 返回所有满足匹配条件的结果,放在列表里
['a', 'b', 'a', 'b']
>>> kk = re.compile(r'\d+')
>>> kk.findall('one1two2three3four4')
['1', '2', '3', '4']
>>> re.findall(kk,"one1234")
['1234']
```

# 2. 正则表达式

- re模块

（5）sub()和subn()函数

功能：这两个函数都是用来实现搜索并替换功能，都是将某个字符串中所有匹配正则表达式的部分进行某种形式的替换。sub()函数返回一个用来替换的字符串，subn()函数还可以返回一个表示替换的总数，替换后的字符串和表示替换总数的数字一起作为一个拥有两个元素的元组返回。

语法格式：

re. sub (pattern, repl, string)

re. subn (pattern, repl, string)

# 2. 正则表达式

- 示例

```
>>> str = "hello 123 world 456"
>>> print(re.sub('\d+','222',str))
  #将串str中所有数字串均替换成了'222'
hello 222 world 222
>>> print(re.subn('\d+','222',str))
('hello 222 world 222', 2)
```

# 2. 正则表达式

- re模块

（6）split()函数

re模块的split()方法与字符串的split()方法相似，前者是根据正则表达式分割字符串，相比后者显著提升了字符分割能力。

语法格式：

re. split(pattern, string)

# 2. 正则表达式

- 示例

```
import re
str = 'aaa bbb ccc;ddd\teee,fff'
print(str)
print(re.split(r';',str))        #单字符分割
print(re.split(r'[;,]',str))     #两个字符以上切割需要放在 [ ] 中
print(re.split(r'[;,\s]',str)) #增加空白字符切割
```

结果：

```
aaa bbb ccc;ddd      eee,fff
['aaa bbb ccc', 'ddd\teee,fff']
['aaa bbb ccc', 'ddd\teee', 'fff']
['aaa', 'bbb', 'ccc', 'ddd', 'eee', 'fff']
```

# 小 结

- 字符串：介绍了字符串的创建、基本操作、函数和方法；
- 正则表达式：介绍了正则表达式的概念、构造方法、re模块。